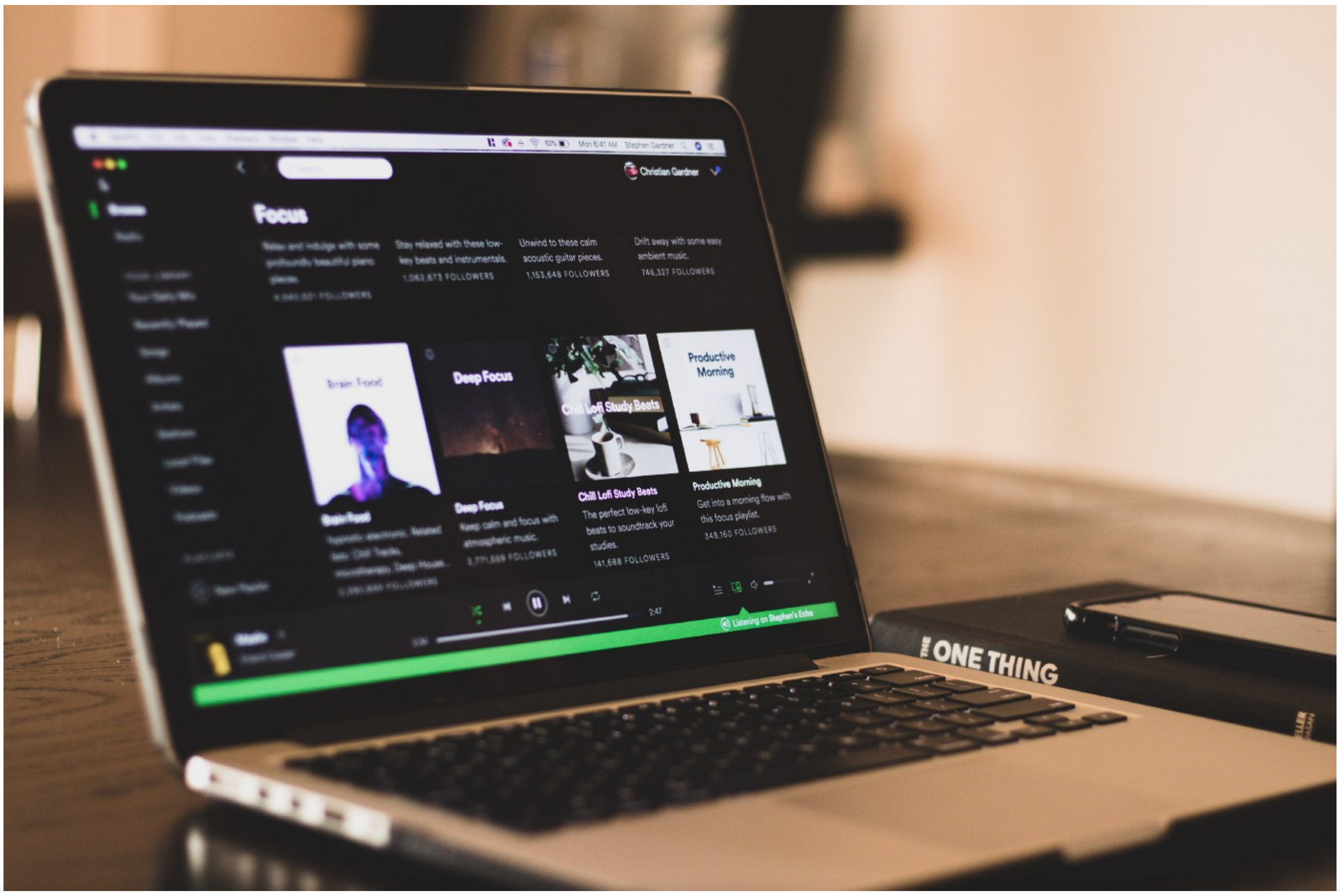
Published in Towards Data Science

Cameron Watts Follow

Dec 17, 2021 · 7 min read · Listen

Extracting Song Data From the Spotify API Using Python

Taking advantage of the data Spotify keeps on its library, and using this for our machine learning projects



This article is the first in a four-part series of articles showcasing our work building a music recommendation system, using Spotify's million playlist dataset [1]. This article details the extraction of data from Spotify's API, from the unique song identifiers that make up the dataset. The other articles in this series are as follows:

- Part I: (This article)
- Part II: EDA and Clustering
- Part III: Building a Song Recommendation System with Spotify
- Part IV: Deploying a Spotify Recommendation Model with Flask

Introduction

Spotify keeps a lot of data on its songs internally, that we can access through the Spotify API. The Spotify API is a great public tool, allowing the use of Spotify's wealth of data on music to build many kinds of systems. In this article, we learn to use this API through Python's Spotipy package to extract data from unique song identifiers.

The imports we need for this project are as follows:

```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
```

What Can We Do With the Spotify API?

The Spotify API is quite powerful, and gives us access to a lot of information about any song or artist on Spotify. This ranges from features describing the "feel" of the audio, such as the variables "liveness", "acousticness", and "energy", through to the features describing the popularity of the artist and song. We can also get more advanced information from this API, such as the predicted position of each beat in the song, if we want to do a more advanced analysis of the data.

Other Spotify features, such as the recommendation engine and search are also available through the Spotify API. To learn more about the Web-API that the Spotipy

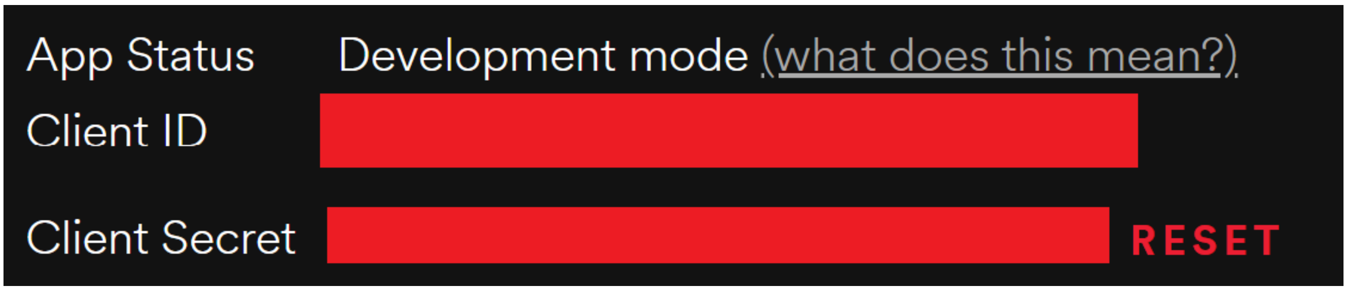
What Will This Article Cover?

This article will cover the basics of using the Spotify web API through Spotipy. This ranges from getting access tokens and authentication, through to extracting features from songs in a playlist, given its associated URI (Uniform Resource Identifier).

API Keys

If you haven't used an API before, the use of various keys for authentication, and the sending of requests can prove to be a bit daunting. The first thing we'll look at is getting keys to use. For this, we need a Spotify for developers [2] account. This is the same as a Spotify account, and doesn't require Spotify Premium. From here, go to the dashboard and "create an app". Now, we can access a public and private key, needed to use the API.

Now that we have an app, we can get a client ID and a client secret for this app. Both of these will be required to authenticate with the Spotify web API for our application, and can be thought of as a kind of username and password for the application. It is best practice not to share either of these, but especially don't share the client secret key. To prevent this, we can keep it in a separate file, which, if you're using Git for version control, should be Gitignored.



App Status Development mode (what does this mean?)
Client ID [redacted]
Client Secret [redacted] RESET

These are the respective Client ID and Client Secret keys, which are needed to authenticate our app. Image by Author

Authenticating with Spotipy

There are two types of authentication that we can perform with the Spotipy library. Firstly, we can authenticate without a specific user in mind. This allows us to access general features of Spotify, and see playlists. Without this, we cannot see stats specific to a user, such as their following lists, and stats of music listened to.

```
#Authentication - without user
client_credentials_manager = SpotifyClientCredentials(client_id=cid,
client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager =
client_credentials_manager)
```

To authenticate with an account, we need to prompt a user to sign in. This is done using the “prompt_for_user_token” method in the “spotipy.utils” section of the package. As we do not use this for this project, this won’t be explored, but more can be read about this in the documentation for the Spotipy package [3].

Using the Spotify Object

Both types of authentication create the same Spotify object, just with different methods of creation. This means that the same class methods are usable for either method of authentication, with the exception of those relating to the current user. Now, using this object, we can interact with the Spotify API, to get the information that we want.

What is a URI Anyway?

An important component of using the Spotify API is the use of the uniform resource identifiers, pointing at each object in the API. We need a URI to perform any function with the API referring to an object in Spotify. The URI of any Spotify object is contained in its shareable link. For example, the link to the Global top songs playlist, when found from the Spotify desktop application, is:

> “https://open.spotify.com/playlist/37i9dQZEVXbNG2KDcFcKOF?si=77d8f5cd51cd478d”

The URI contained in this link is “37i9dQZEVXbNG2KDcFcKOF” — if we use this with the API then we will be referencing the Global top songs playlist. You may also see the URI listed in the format “spotify:object_type:uri”, which also works, and if anything is a more valid way of referring to the object. Using these URIs, we will extract features of songs in a playlist, and in turn extract a series of features from these songs, such that

The first method that we will use in extracting features from tracks in a playlist is the "playlist_tracks" method. This method takes the URI from a playlist, and outputs JSON data containing all of the information about this playlist. Luckily, the Spotipy package decodes this for us, so we can parse through this data fairly easily and Pythonically.

We want to extract the track data here, such that we can get features from this. This can be done through the following section of code, which extracts the URI for each song in the playlist given (still the global top 40 for our example):

```
playlist_link =
"https://open.spotify.com/playlist/37i9dQZEVXbNG2KDcFcKOF?
si=1333723a6eff4b7f"
playlist_URI = playlist_link.split("/")[-1].split("?")[0]
track_uris = [x["track"]["uri"] for x in
sp.playlist_tracks(playlist_URI)["items"]]
```

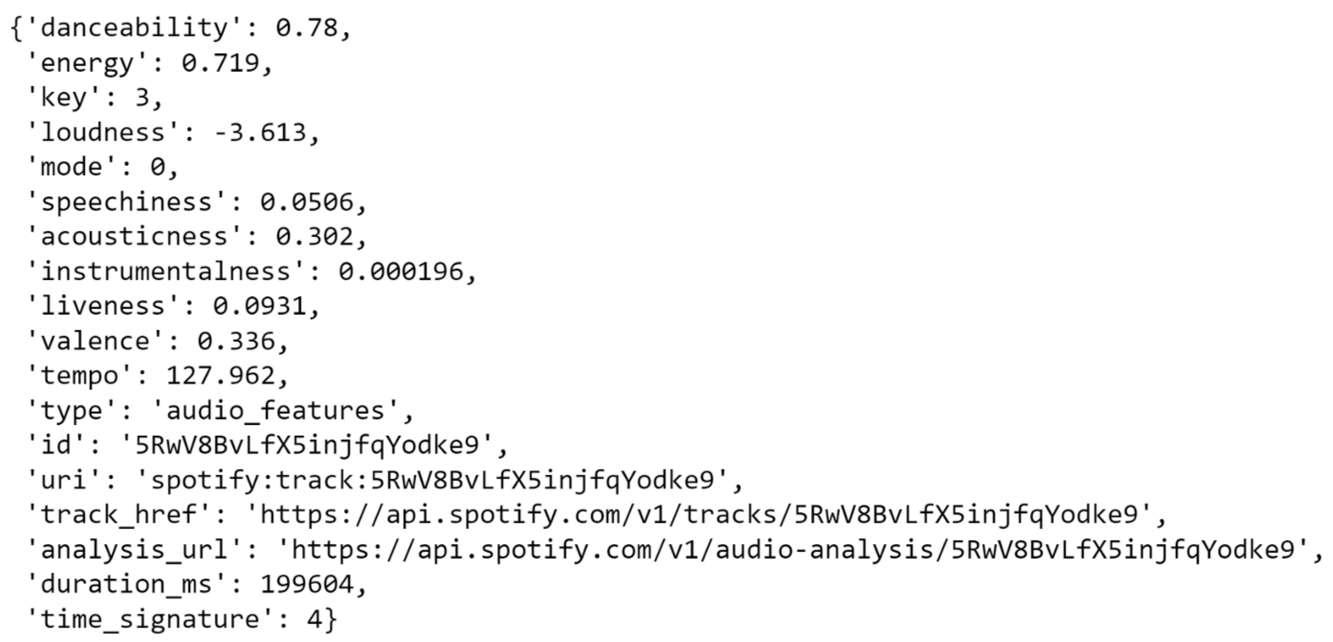
While we're here, we can also extract the name of each track, the name of the album that it belongs to, and the popularity of the track (which we expect to be high in this case — we're looking at the most popular songs globally). From the artist, we can find a genre (though not airtight — artists can make songs in multiple genres), and an artist popularity score.

```
for track in sp.playlist_tracks(playlist_URI)["items"]:
    #URI
    track_uri = track["track"]["uri"]

    #Track name
    track_name = track["track"]["name"]

    #Main Artist
    artist_uri = track["track"]["artists"][0]["uri"]
    artist_info = sp.artist(artist_uri)

    #Name, popularity, genre
    artist_name = track["track"]["artists"][0]["name"]
    artist_pop = artist_info["popularity"]
    artist_genres = artist_info["genres"]

    #Album
```

Extracting Features from Tracks

Now that we have a list of track URIs, we can extract features from these tracks, in order to perform our analysis. Spotify has a list of these features for each of its tracks, from analysis of the audio. We can access these with a single method of the spotify object `audio_features(uri)`. This gives us a list of mostly numerical features that we can use for our analysis.

```
sp.audio_features(track_uri)[0]
```



```
{'danceability': 0.78,
 'energy': 0.719,
 'key': 3,
 'loudness': -3.613,
 'mode': 0,
 'speechiness': 0.0506,
 'acousticness': 0.302,
 'instrumentalness': 0.000196,
 'liveness': 0.0931,
 'valence': 0.336,
 'tempo': 127.962,
 'type': 'audio_features',
 'id': '5RwV8BvLfX5injfqYodke9',
 'uri': 'spotify:track:5RwV8BvLfX5injfqYodke9',
 'track_href': 'https://api.spotify.com/v1/tracks/5RwV8BvLfX5injfqYodke9',
 'analysis_url': 'https://api.spotify.com/v1/audio-analysis/5RwV8BvLfX5injfqYodke9',
 'duration_ms': 199604,
 'time_signature': 4}
```

Results from the above code. Image by Author.

Other Uses

There are plenty of other things that you can do with this object, including building and editing playlists, controlling your own Spotify playback, and accessing many different aspects of objects in Spotify. We've only covered a small portion of these in this article, but you can read more in the documentation for the Spotipy package, here [3].

Usage For This Project

In this project, the Spotify API is used to extract a set of features (the ones showcased linked Github repository for this project, we use a script to write a function for this, returning a list of features given the URI for a track.

Conclusion

Spotify keeps a lot of internal data, and allows us to access it through their API. This is extremely useful when we want to use our own data to build datasets for analysis. In the million playlist dataset [1], it is extremely useful to be able to extract features about the contained songs, such that we can better understand how songs relate to each other, and perform clustering to build our own recommendation engine.

Again, this article is part 1 of a series in which we built a recommendation engine using Spotify's million playlist dataset. The other articles in this series are linked below:

- Part I: (This article)
- Part II: EDA and Clustering
- Part III: Building a Song Recommendation System with Spotify
- Part IV: Deploying a Spotify Recommendation Model with Flask

In future articles, we will explore the dataset, and create a clustering-based recommendation model based on the features extracted.

The Github repository for this project is linked here:

https://github.com/enjuichang/PracticalDataScience-ENCA

References / Further Reading

[1] Spotify / AICrowd, Million Playlist Dataset (2018), https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge

[2] Spotify, Spotify for Developers, https://developer.spotify.com/

[3] plamere, Spotipy documentation, https://spotipy.readthedocs.io/en/2.19.0/

[4] plamere, Spotipy Codebase, https://github.com/plamere/spotipy

Open in app Get started